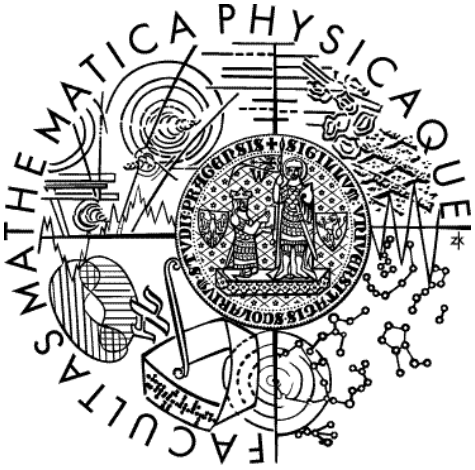
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Matej Dioszegi

Implementácia moderných prístupov v elektronickom zdravotnom zázname

Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Miroslav Nagy
Studijní program: Informatika
2007

Chcel by som poďakovať mojim rodičom za podporu počas celého štúdia, vedúcemu tejto práce Mgr. Miroslavovi Nagyovi za všetky konzultácie týkajúce sa elektronického zdravotného záznamu a za všetky rady poskytnuté počas písania.
Takisto by som chcel poďakovať Prof. RNDr. Jane Zvárovej DrSc. za poskytnutie technických prostriedkov pre vývoj a testovanie.

Prehlasujem, že svoju bakalársku prácu som napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičiavaním práce.

V Prahe dňa 7.8.2007 Matej Dioszegi

Obsah

Názov práce: Implementácia moderných prístupov v elektronickom zdravotnom zázname
Autor: Matej Dioszegi
Katedra: Katedra softwarového inženýrství
Vedúci bakalárskej práce: Mgr. Miroslav Nagy
e-mail vedúceho: nagy@euromise.cz

Abstrakt: V predloženej práci sa študujú možnosti implementácie elektronického zdravotného záznamu pomocou moderných technológií. Hlavným cieľom je väčšia dekompozícia serverovej časti implementácie EHR v porovnaní s minulými verziami. Je použitá myšlienka webových služieb, pri ktorej každá časť aplikácie poskytuje ostatným vzdialené služby. Jednotlivé komponenty medzi sebou komunikujú pomocou protokolu SOAP. Takisto sa študuje možnosť separovať administratívne údaje o pacientoch do samostatného skladišťa, ktoré sa na tento účel hodí viac ako dátové stromy, ktoré sú použité na uloženie klinických údajov. Toto oddelenie údajov je vhodné aj z pohľadu anonymizácie dát pri štatistickom spracovaní. V práci ostáva zachovaný spôsob uloženia pacientových klinických dát v dátových stromoch a možnosť flexibilného modelovania dátovej vrstvy pomocou znalostných stromov združených v znalostnej báze.

Kľúčové slová: elektronický zdravotný záznam, znalostná báza, centralizovaná správa pacientov

Title: Implementation of modern approaches in electronic health record
Author: Matej Dioszegi
Department: Department of software engineering
Supervisor: Mgr. Miroslav Nagy
Supervisor´s e-mail address: nagy@euromise.cz

Abstract: In the present work we study the possible implementations of electronic health record using modern technologies. The main goal is decomposition of server functionality among multiple parts. We use the idea of web-services, which means that each part of the system offers services available for remote hosts. The components communicate with each other using the SOAP protocol. We also study the possibility of storing patient's administrative information into separate storage, which suits the purpose better than original data trees, which are used to store clicnical information. Besides, this separation is suitable for anonymization of data used by statistical computing. In my work the manner of storing patient´s clinical data in data trees and possibility of flexible modeling of data layer using knowledge trees, which together create a knowledge base, is preserved.

Keywords: electronic health record, knowledge base, centralized patient management

Kapitola 1

Úvod

Na úvod práce je vhodné priblížiť čitateľovi význam pojmu „elektronický zdravotný záznam“, ktorého sa táto práca týka.

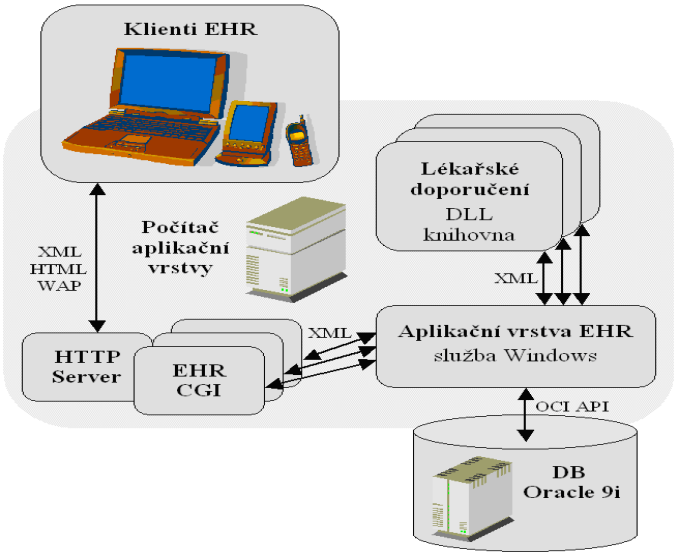
Voľným citovaním normy ČSN ENV 13606 a [1] môžeme elektronický zdravotný záznam (electronic health record, skratka EHR) charakterizovať nasledovne: „Zdravotné záznamy sú v súčasnosti zavedenou súčasťou klinickej praxe. Tieto záznamy obsahujú dôležité informácie pre liečebnú starostlivosť a používajú sa rôznymi spôsobmi na rozdielne účely. Snahou je reprezentovať tieto záznamy na elektronickom médiu tak, aby ich bolo možné spracovať počítačovým systémom. Elektronický zdravotný záznam je potom technologickým prostriedkom pre dokumentovanie postupu starostlivosti poskytovanej jedincovi.“

Podrobnejšie sa definíciám pojmov z tejto oblasti a súčasnému stavu EHR v ČR venuje [1].

1.1 Pôvodné implementácie a ich nedostatky

V roku 2002 bola na Ústave informatiky AV ČR napísaná pilotná implementácia servera elektronického zdravotného záznamu, ktorá je popísaná v [1]. Autor pri jej vytváraní vyvinul flexibilný model dátovej vrstvy založený na znalostných stromoch združených v znalostnej báze a dátových stromoch, ktorých uzly obsahujú hodnoty a odkazujú sa do znalostnej bázy.

Pri nasadzovaní do praxe sa najprv s odborníkmi na medicínu namodeluje znalostná báza, ktorá popisuje zbierané dáta a táto znalostná báza sa uloží do databázy. Lekár potom pri práci s pacientom narába s pacientovým dátovým stromom, do ktorého zapisuje, prípadne edituje, hodnoty. Tu sa stretávame s prvým problémom, a síce že korene dátových stromov nie sú žiadnym spôsobom zviazané so základnými identifikačnými údajmi o pacientovi – menom, priezviskom a rodným číslom.



Obrázok 1: Architektúra pôvodnej implementácie EHR

Ďalším nedostatkom je neexistencia dátového typu „dátum“, ktorý bol modelovaný pomocou logickej hodnoty s istou platnosťou (ňou sa rozumie interval, počas ktorého sa údaj považuje za platný). Dolná hranica intervalu je povinná a považovaná za hodnotu dátumu.

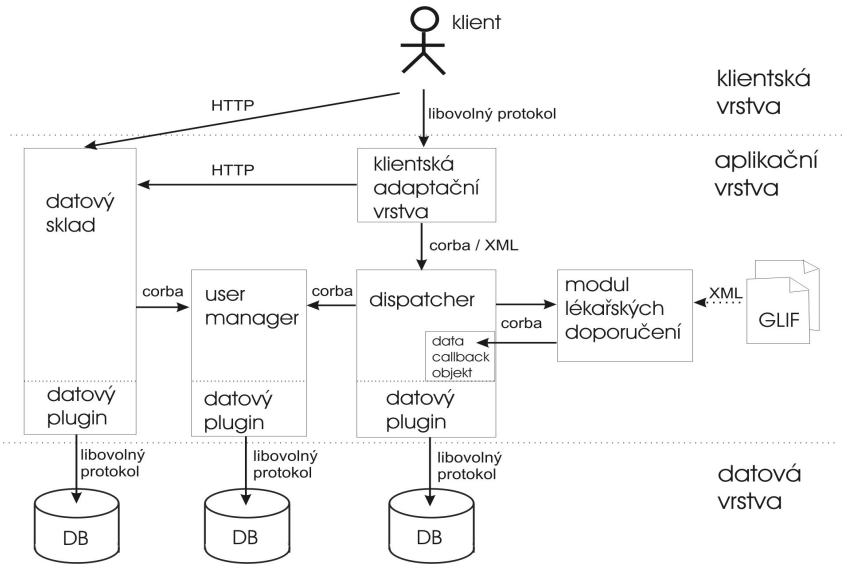
Náčrt tejto implementácie je na obrázku 1, prevzatý z [1].

Pôvodná implementácia vyžadovala, aby všetky komponenty systému bežali na platforme Win32. Rozšíriteľnosť systému bola založená na DLL knižniciach, čo systém ešte pevnejšie zviazalo s menovanou platformou. Preto sa v roku 2004 vytvorila novšia implementácia, popísaná v [2]. Táto je z veľkej časti zameraná na formalizáciu lekárskych doporučení. Hlavnou zmenou je rozloženie systému na viac komponentov, ktoré medzi sebou komunikujú pomocou middlewaru CORBA. V tejto implementácii bol navrhnutý a naimplementovaný spôsob vyhodnocovania lekárskych doporučení formalizovaných podľa štandardu GLIF. Predkladaná práca sa však lekárskym doporučeniam nebude venovať, nakoľko táto téma nie je triviálna a je nad rámec tejto práce. Pôvodne existovala možnosť prevziať modul na výpočet lekárskych doporučení z druhej verzie, ale testovaním sa prišlo na to, že tento modul nie je dostatočne stabilný.

K nedostatkom druhej verzie patrí znova absencia jednoznačného zviazania dátových stromov s konkrétnymi pacientmi, absencia typu dátum a

technické problémy s klientskou adaptačnou vrstvou a použitým middleware.

Náčrt fungovania druhej implementácie je na obrázku 2, prevzatý z [2].



Obrázok 2: Architektúra MUDR-II

Kapitola 2

Nová implementácia a jej zmeny oproti predošlým verziám

Predkladaná práca sa venuje implementácii aplikačnej vrstvy s použitím moderných prístupov. Za pracovný názov bol zvolený MUDRj, ktorý sa bude v ďalšom texte používať. Pojem „moderné prístupy“ je dosť obecný a preto je potrebné vysvetliť, čo všetko zahŕňa.

V prvom rade ide o použitie viacerých komponentov, ktoré môžu byť distribuované na viacero uzlov (počítačov). Rozdelenie na komponenty v [2] je trochu zložité. V dnešnej dobe klientská adaptačná vrstva stráca zmysel, pretože server EHR beží v kontajneri aplikačného servera, kde prijíma od klientov príkazy a odpovedá na ne. Komunikácia oboma smermi používa protokol HTTP.

Moderným prístupom je implementácia jednotlivých komponentov ako tzv. web-services. Táto verzia je prehľadnejšia ako implementácia s použitím CORBy. Navyše existuje kvalitný engine pre webové služby – Axis2 [3] od Apache Software Foundation (http://www.apache.org). Každý komponent poskytuje nejaké rozhranie, prostredníctvom ktorého sa na ňom dajú vzdialene volať procedúry a funkcie. Toto rozhranie je definovane vo formáte WSDL (Web Services Definition Language), čo je XML dokument popisujúci operácie, ktoré služba poskytuje. Viac sa o Axis-e čitateľ dozvie v kapitole 3.

Za moderný prístup k EHR sa považuje aj oddelenie administratívnych údajov pacientov od znalostí klinických. Toto oddelenie je vhodné z viacerých dôvodov, medzi ktoré patrí napríklad anonymizácia dát pre štatistické spracovanie, zrýchlené vyhľadávanie pacientov podľa ich kontaktných údajov a samozrejme aj zvýšená bezpečnosť osobných údajov, ktoré vďaka tomuto oddeleniu môžu byť fyzicky uložené na inom mieste ako samotný server EHR. Nasledovná komunikácia medzi serverom EHR a úložiskom osobných údajov musí samozrejme prebiehať niektorým z bezpečných spôsobov (použitie SSL, IPSec …), čo je aj v tejto práci implementované.

2.1 Rozšírenie komunikačného protokolu

Do komunikačného protokolu medzi EHR serverom a klientom boli pridané príkazy na prácu s administratívnymi údajmi pacienta, potom príkaz `<get_data_to_sign>`, ktorým si užívateľ vypýta dáta, ktoré chce podpísať a `<enter_signed_data>`, ktorým užívateľ vloží podpísané dáta. Dátová vrstva sa rozrástla o spoločnú tabuľku `EHR_DATA_MULTIMEDIA`, kde sa ukladajú multimédia a o dátové typy „dátum“ a „LDAP referencia“. Bližšie sa im budeme venovať pri popise jednotlivých komponentov.

2.2 Centralizovaná správa pacientov

Jedným z cieľov práce bolo navrhnúť spôsob uloženia administratívnych informácií o pacientovi do oddeleného skladu, ktorý je určený špeciálne pre tento účel. Prvým krokom bolo zistiť, čo by sa ako skladisko dalo použiť. Možnosti sú:

1. Relačná databáza – toto riešenie pripadlo v úvahu ako prvé. Bolo by potrebné navrhnúť schému v akom tvare budú administratívne údaje uložené. Relačná databáza je vhodná pre uloženie všeobecných dát, preto by bolo potrebné vytvárať logickú štruktúru vložených administratívnych údajov, čo sa javilo ako nevýhoda.
2. Adresárový server – tento spôsob sa dá predstaviť ako telefónny zoznam, v ktorom sú uložené „vizitky“ pacientov. Každá vizitka má nejakú vopred stanovenú štruktúru. Toto riešenie bolo v porovnaní s relačnou databázou vhodnejšie, pretože stromová štruktúra, ktorá je pre administratívne údaje potrebná, je podporovaná serverom natívne.

V súčasnej implementácii bola použitá druhá možnosť – adresárový server a to z vyššie uvedených dôvodov.

Adresár sa chápe ako množina informácií s podobnými atribútmi, ktoré sú usporiadané logickým spôsobom do stromov. Strom často odráža nejakú geografickú, politickú alebo organizačnú štruktúru. Koreňom stromu je uzol, ktorého význam pokrýva celý strom. Postupne s klesaním do nižších vrstiev sa dostávame k presnejším informáciám.

K službám adresárových serverov sa pristupuje protokolom LDAP (Lightweight Directory Access Protocol), ktorý vznikol odľahčením protokolu DAP (Directory Access Protocol).

Každý uzol v strome má tzv. *distinguished name* (DN) a *relative distinguished name* (RDN). RDN identifikuje záznam medzi jeho súrodencami. To znamená, že viacero uzlov môže mať rovnaké RDN, ale tieto uzly nemôžu mať spoločného otca. DN identifikuje záznam

jednoznačne v celom strome. Pozostáva z RDN uzlov od záznamu až po koreň stromu, oddelených od seba čiarkou. Konkrétny príklad DN z adresárového servera, ktorý je súčasťou priloženého riešenia:

```
uid=6fbd39e975c9265f52d384b211e99083,ou=patients,o=patients
```

Na tomto príklade je vidieť, že koreňom stromu je uzol, ktorého RDN je „o=patients“. Pod ním je uzol s RDN „ou=patients“, ktorý obsahuje jednotlivých pacientov. Každý pacient má atribút uid, ktorého hodnota sa vypočíta zahashovaním mena, priezviska a rodného čísla. Tento atribút spolu s hodnotou tvorí RDN jedného pacienta.

Okrem uzlu s RDN „ou=patients“ sa pod koreňom nachádza aj uzol s RDN „ou=insuranceCompanies“, ktorý obsahuje zoznam zdravotných poisťovní. Na položky z tohto zoznamu sa odkazuje atribút pacienta, ktorý obsahuje informáciu o poisťovni, v ktorej je pacient poistený. Viac sa komponentu na centralizovanú správu pacientov budeme venovať v kapitole 3.4.

2.3 Architektúra MUDRj

Architektúra MUDRj bola z veľkej časti inšpirovaná architektúrou MUDRII, ktorá je popísaná v [2].

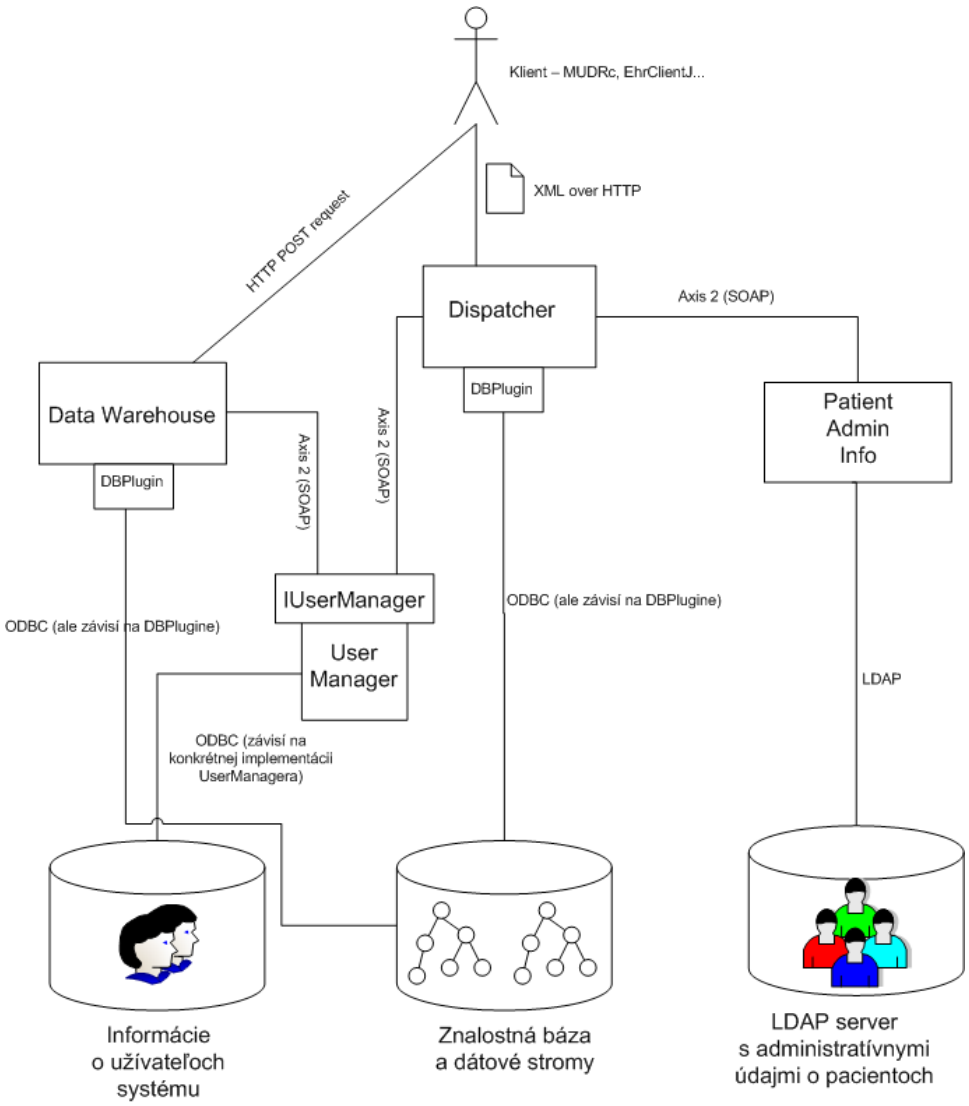
Zachováva komponenty `Dispatcher`, `UserManager` a `Data Warehouse` (dátový sklad). `Dispatcher` slúži na príjem požiadaviek a odosielanie odpovedí klientom. Takisto pomocou databázového pluginu vykonáva operácie v dátovej vrstve. Komponent `UserManager` má na starosti užívateľov systému. Poskytuje funkcie na autentifikáciu užívateľov, ich pridávanie, editovanie a získavanie informácií o nich. Komponent `Data Warehouse` slúži pre nahrávanie súborov do databázy.

Novo pridaný komponent `PatientAdminInfo` poskytuje administratívne informácie o pacientoch, ktoré získava z LDAP serveru. Klientská adaptačná vrstva sa v tejto implementácii nevyskytuje a modul pre výpočet lekárskych doporučení môže byť doplnený neskôr, ideálne ako webová služba.

Komponenty medzi sebou komunikujú protokolom SOAP [5]. O vzájomnú komunikáciu sa stará už spomínaný Axis2. Spojenie s databázou je na obrázku 3 zakreslené ako ODBC, ale závisí na databázovom plugine, ktorý komponent využíva. Konkrétne v tejto práci je použitá databáza Oracle 10g [15], takže ako dátový plugin je použitá trieda `EHROraService`, ktorá implementuje rozhranie `DBPlugin`. Podobne by sa dali naimplementovať pluginy pre iné databázy (MS SQL Server…). Následne stačí v konfiguračnom súbore prepísať riadok s menom triedy implementujúcej `DBPlugin` a komponent sa bude pripájať na inú databázu. Tým je zabezpečená nezávislosť na použitom databázovom stroji.

Komunikácia medzi komponentom `PatientAdminInfo` a adresárovým serverom prebieha na protokole LDAP.

Schéma fungovania súčasnej implementácie je na obrázku 3.



Obrázok 3: Architektúra MUDRj

Kapitola 3

Popis implementácie servera elektronického zdravotného záznamu

V tejto kapitole budú podrobnejšie popísané jednotlivé časti súčasného riešenia. Najprv ale bude technicky upresnené, ako časti medzi sebou spolupracujú.

3.1 Komunikácia medzi komponentmi

Ako už bolo spomenuté, komponenty na vzájomnú komunikáciu používajú protokol SOAP. Samotný prenos a prácu so SOAP správami zabezpečuje nástroj Axis2. Tento nástroj môže bežať ako samostatný server, alebo ako aplikácia na nejakom aplikačnom serveri. Bola zvolená druhá možnosť, ktorú odporúčajú aj autori Axis-u a ako aplikačný server bol použitý Tomcat 6.0 [4], ktorý takisto pochádza z dielne Apache Software Foundation. Táto verzia Tomcata potrebuje k svojmu fungovaniu JRE[1] v1.5 a vyššiu (pre samotný Axis2 stačí JRE 1.4).

Inštalačný adresár Axis-u obsahuje adresár services, do ktorého sa umiestňujú .war archívy s webovými službami. Archív s webovou službou má nasledujúcu adresárovú štruktúru:

```
- MyService
   - META-INF
      -services.xml
   - lib
   - com
      -myPackage
         -ClassFrompackage.class
         -Class2fromPackage.class
      -myOtherPackage
         -OtherClass.class
```

`Services.xml` slúži na popis webovej služby. Obsahuje meno a popis webovej služby, mená tried, ktoré implementujú prijímanie a odosielanie SOAP správ, parameter `ServiceClass` a ďalšie elementy. Dôležitý je

1 Java Runtime Enviroment – virtuálny stroj jazyka Java

parameter `ServiceClass`. Pri spúšťaní služby sa vytvorí inštancia uvedenej triedy.

Na základe verejných metód `ServiceClass` sa vygeneruje WSDL. Toto WSDL je možné prezrieť priamo vo webovom prehliadači, keď sa do riadka adresy zadá za názov služby `?wsdl`. Napríklad ak Tomcat a Axis bežia na localhost na porte 8080, tak po zadaní

```
http://localhost:8080/axis2/services/patientAdminService?wsdl
```

sa objaví WSDL pre uvedenú službu.

3.1.1 WSDL

Skratka znamená Web Services Description Language [7]. Je to jazyk na popis webových služieb, založený na XML. WSDL popisuje webovú službu pomocou týchto elementov:

a. <portType> operácie, ktoré služba vykonáva

b. <message> správy, ktoré používa

c. <types> dátové typy používané v komunikácii

d. <binding> komunikačné protokoly, ktoré služba používa.

Tieto elementy sú uzavreté v koreňovom elemente <definitions>.

Z tradičného programátorského pohľadu sa `portType` dá prirovnať k triede alebo modulu, ktoré majú nejaké verejné rozhranie. Operácie, ktoré služba vykonáva, môžeme rozdeliť do 4 skupín (tak ich definuje aj WSDL):

1. one-way – nevracajú žiaden výsledok, ekvivalent procedúry
2. request-response – operácie vracajúce odpoveď na nejakú požiadavku, ekvivalent funkcie
3. solicit-response – operácia vyšle požiadavku a čaká na odpoveď
4. notification – operácia vyšle požiadavku, ale nečaká odpoveď.

Najčastejším prípadom sú request-response a one-way operácie. Solicit-response môže slúžiť napríklad na zisťovanie prítomnosti klienta a notification napríklad na ohlasovanie štartu servera.

Element `binding` definuje formát správ, ktoré tvoria vstup a výstup operácií a detaily protokolu pre webovú službu. Element má dva atribúty a to `name`, čo je meno binding-u a môže byť ľubovoľné. Atribút `type` odkazuje na „port“ tohto bindingu (musí byť zhodný s `name` atribútom niektorého elementu `portType`). Binding obsahuje práve jeden element, ktorý už závisí od konkrétneho protokolu, napríklad `soap:binding` alebo

`http:binding` a následne pre každú operáciu element `operation`, kde je uvedený formát jej vstupu a výstupu.

Element `types` popisuje pomocou XmlSchema [6] dátové typy používané v komunikácii. Elementov `messagge` je viac a každý z nich zaobaľuje nejaký dátový typ z elementu `types`.

Pre prehľadnosť budú uvedené časti z WSDL, ktoré vygeneroval Axis pre službu PatientAdminService. Kompletné vygenerované WSDL sú priložené pri jednotlivých komponentoch na CD.

```
<xs:element name="addPatientContact">
       <xs:complexType>
              <xs:sequence>
                     <xs:element name="patientId"
                     nillable="true" type="xs:string"/>
                     <xs:element name="c" nillable="true"
              type="ns1:Contact"/>
              </xs:sequence>
       </xs:complexType>
</xs:element>
```

Uvedený element s názvom `addPatientContact` popisuje prvok, ktorý obsahuje atribúty „patientId“ typu xs:string a „c“, ktorý je typu Contact. Na tento element, ktorý pochádza z elementu `types`, sa odkazuje element `message`:

```
<wsdl:message name="addPatientContactMessage">
   <wsdl:part name="part1" element="ns0:addPatientContact"/>
</wsdl:message>
```

Ten už hovorí, že správu menom `addPatientContactMessage` tvorí jeden element - `addPatientContact`. Takto definovaná správa sa použije v elemente `port`, pri popise operácie:

```
<wsdl:operation name="addPatientContact">
       <wsdl:input message="axis2:addPatientContactMessage"
              wsaw:Action="urn:addPatientContact"/>
       <wsdl:output message="axis2:addPatientContactResponse"/>
</wsdl:operation>
```

Týmto elementom sa zadefinovalo, že webová služba poskytuje operáciu addPatientContact, ktorá na vstupe očakáva správu "`addPatientContactMessage`" (element wsdl:input), na ktorú odpovie správou "`addPatientContactResponse`" (element wsdl:output). Atribút `wsaw:Action` slúži na vygenerovanie správnej „Action“ hlavičky v SOAP správe (v našom prípade je tu meno funkcie zo

`ServiceClass`, ktorá je zavolaná po prijatí vstupnej správy operácie addPatientContact).

Správa `addPatientContactResponse` je definovaná nasledovne:

```
<wsdl:message name="addPatientContactResponse">
<wsdl:part name="part1" element="ns0:addPatientContactResponse"/>
</wsdl:message>
```

Odkazuje sa na element menom `addPatientContactResponse`, ktorý je zadefinovaný v elemente `types` takto:

```
<xs:element name="addPatientContactResponse">
       <xs:complexType>
              <xs:sequence>
              <xs:element name="return" nillable="true"
              type="xs:string"/>
              </xs:sequence>
       </xs:complexType>
</xs:element>
```

Z pohľadu klienta, to vyzerá tak, že zavolá vzdialenú procedúru addPatientContact (prostredníctvom triedy RemotePatientAdmin, ktorá vzdialený komponent reprezentuje a vystupuje voči nemu v úlohe klienta), ktorej na vstupe poskytne ID pacienta a inštanciu triedy Contact a dozvie sa DN2 pridaného kontaktu. V prípade že sa kontakt vložiť nepodarí, klientovi sa vráti hodnota null. Pre úplnosť ešte definícia typu `Contact` z WSDL:

```
<xs:complexType name="Contact">
<xs:sequence>
   <xs:element name="city" nillable="true" type="xs:string"/>
   <xs:element name="dn" nillable="true" type="xs:string"/>
   <xs:element name="email" nillable="true" type="xs:string"/>
   <xs:element name="pobox" nillable="true" type="xs:string"/>
   <xs:element name="postalCode" nillable="true" type="xs:string"/>
   <xs:element name="state" nillable="true" type="xs:string"/>
   <xs:element name="street" nillable="true" type="xs:string"/>
   <xs:element name="telNum" nillable="true" type="xs:string"/>
   <xs:element name="type" nillable="true" type="xs:string"/>
</xs:sequence>
</xs:complexType>
```

Po definícii operácie je potrebné povedať, akým protokolom a v akom formáte sa budú prenášať dáta (parametre a návratové hodnoty). Tieto informácie sa nachádzajú v spomenutom elemente `binding`:

```
<wsdl:binding name="patientAdminServiceHttpBinding"
type="axis2:patientAdminServicePortType">
<http:binding verb="POST"/>
<wsdl:operation name="addPatientContact">
       <http:operation location="addPatientContact"/>
       <wsdl:input>
              <mime:content type="text/xml"/>
```

```
        </wsdl:input>
        <wsdl:output>
                <mime:content type="text/xml"/>
        </wsdl:output>
</wsdl:operation>
```

Týmto je zadefinované viazanie operácie `addPatientContact` na protokol HTTP. Na vstupe prijíma dáta vo formáte text/xml a v tom istom formáte ich aj posiela naspäť. Binding-ov pre službu popisovanú vo WSDL dokumente môže byť zadefinovaných viacero. Axis2 vygeneruje okrem HTTP aj binding pre protokoly SOAP1.1 a SOAP 1.2 a závisí na klientovi služby, ktorý spôsob prenosu si vyberie.

3.2 Dátová vrstva

Schéma dátovej vrstvy je na obrázku v prílohe 2. Oproti verzii v [1] a [2] nastalo niekoľko zmien. Prvou z nich bolo zavedenie dvoch nových dátových typov – LDAP referencia a dátum. To znamenalo vytvorenie tabuliek EHR_DATA_LDAP_REFS a EHR_DATA_DATES, ktoré obsahujú odkaz do tabuľky dátových uzlov a samotnú hodnotu. Dátum sa ukladá vo formáte YYYY-MM-DDTHH:MM:SS[2], preto je typ stĺpca VARCHAR2 a nie Date. Ďalej pribudla tabuľka EHR_DATA_MULTIMEDIA v ktorej sa ukladajú multimediálne dáta vo forme BLOBov.

Kvôli požiadavke na možnosť podpisovania údajov bolo nutné vytvoriť tabuľku EHR_SIGNED_DATA. V nej sa ukladá niekoľko hodnôt. V prvom rade „čisté“ dáta, ktoré sú tvorené vyexportovaným podstromom dátového stromu reprezentovaného pomocou XML. Tieto XML dáta sú uložené ako BLOB. Následne ID lekára, ktorý tieto dáta vložil a dátum ich vloženia. Podstatný je stĺpec, do ktorého sa ukladajú podpísané dáta, takisto vo formáte BLOB. Pri nich je dátum, kedy boli dáta podpísané.

3.2.1 Vyhľadávanie pacientov použitím LDAP referencie

Existencia dátového typu LDAP referencia (LDAP_REF) umožňuje rýchle vyhľadávanie dátového stromu konkrétneho pacienta.

Predstavme si, že lekár chce zistiť výšku pacienta Jána Nováka. Prvým krokom je, že sa v tabuľke EHR_DATA_TEXTS nájde hodnota „Ján“, ktorá odkazuje na dátový uzol, ktorý ukazuje na uzol znalostnej bázy s menom FIRSTNAME. To isté sa urobí pre hodnotu „Novák“, ale hľadajú sa dátové uzly odkazujúce na SURNAME. Keďže nie je nič nezvyčajné, že Jánov Novákov bude dosť veľa, tak k identifikácii je potrebné ešte aj rodné číslo pacienta. Takže sa nájdu aj dátové uzly s hodnotou, ktorá zodpovedá rodnému číslu a odkazujú do znalostnej bázy na uzol BIRTHNUM. Takýto

2 Podľa ISO 8601

uzol by mal byť len jeden, ale ani to nemusí byť celkom pravidlom. Každopádne, pravdepodobnosť, že nájdem dvoch Jánov Novákov s rovnakým rodným číslom a nebude to tá istá osoba je tak nízka, že si dovolíme ju zanedbať. Vo výslednej množine uzlov sa nájde taká trojica uzlov, ktoré majú ako svojho najvyššieho predka rovnaký uzol a tým sa našiel koreň pacientovho dátového stromu. Následne sa v získanom dátovom strome nájde príslušný uzol podľa znalostnej bázy a zistí sa jeho hodnota.

Existencia dátového typu LDAP_REF umožňuje zviazať koreň dátového stromu s konkrétnym pacientom. Keď sa totiž zakladá nový pacient funkciou `add_new_patient`, vypočíta sa mu identifikátor a to zahashovaním krstného mena, priezviska a rodného čísla. Následne na to sa v dátovej vrstve založí koreň dátového stromu a priradí sa mu získaná hodnota. Pre kompatibilitu s klientmi, ktorí nevedia narábať s administratívnymi údajmi oddelene, sa v dátovom strome vytvoria uzly ADMINISTRATIVE a pod ním FIRSTNAME, SURNAME a BIRTHNUM s patričnými hodnotami.

V prípade, že lekár hľadá výšku pána Nováka s využitím LDAP referencie, vykonajú sa nasledujúce kroky: Najprv sa od komponentu `PatientAdminInfo` získa na základe mena, priezviska a rodného čísla identifikátor pacienta (ak pacient existuje). Následne sa prehľadá jedna tabuľka (EHR_DATA_LDAP_REFS). V nej sa podľa získaného identifikátora nájde číslo dátového uzlu, ktorý je koreňom dátového stromu pacienta.

Problém pri tomto prístupe môže nastať v prípade, že pacienta založil starý klient. V takomto prípade server nie je schopný vypočítať identifikátor určujúci pacienta, pretože nemá k dispozícii požadované údaje. To vyplýva z vlastnosti komunikačného protokolu, ktorý pre starého klienta poskytuje len funkciu `enter_data_file`. Vytvorenie nového pacienta potom vyzerá ako séria viacerých príkazov `enter_data_file`. Ale v momente vkladania uzlu typu PATIENT (ako koreňa dátového stromu) server nevie, či vôbec nejaké meno a priezvisko vložené bude. Preto do tabuľky EHR_DATA_LDAP_REFS vloží „dummy“ hodnotu. Teoreticky by sa tento problém dal riešiť tak, že pri spracovávaní príkazu `enter_data_file` by sa server pozrel na znalostný uzol, na ktorý vkladaný dátový súbor ukazuje a ak by to bol jeden z FIRSTNAME, SURNAME alebo BIRTHNUM tak by sa mohol pokúsiť vypočítať identifikátor pacienta zo svojho okolia. Tu sa však ukazujú problémy s logickými obmedzeniami dátových stromov, pretože starí klienti pri zakladaní nového pacienta nie sú nútení uviesť jeho rodné číslo (ani žiadny iný atribút). Takisto nie sú nútení ho niekedy počas existencie stromu vôbec zadať.

V súčasnej implementácii sa ráta s používaním funkcie `add_new_patient`. Predpokladá sa úplné odstránenie vetvy ADMIN z dátových stromov a využívanie adresárového servera pre účely ukladania administratívnych informácií pacientov.

3.3 Komponent Dispatcher

Tento komponent je vstupným bodom do aplikácie. Je umiestnený v kontajneri aplikačného serveru a spracováva požiadavky od klientov. Komunikácia medzi EHR serverom a klientom prebieha prostredníctvom XML dokumentov. Ich štruktúra je definovaná v XmlScheme [6] v súbore `EHRSchema.xsd` na priloženom CD.

Prvým krokom po prijatí požiadavky je jej validácia podľa XmlSchemy. Ak ňou prijatá požiadavka prejde, začína sa jej spracovanie. Najprv `Dispatcher` zistí, o akého klienta sa jedná. Súčasní klienti (EHRc, EHRclient, MUDRc) zatiaľ nevedia pracovať s oddelenými administratívnymi údajmi, preto ich požiadavok v koreňovom elemente neobsahuje atribút `schema-version`. Teda ak server nenájde tento atribút, snaží sa klientovi vrátiť odpoveď v tvare v akom ju poskytoval MUDR a MUDRII. To znamená, že prepisuje dátové typy „LDAP referencia“ na typ „Directory“ a „Dátum“ na „Boolean“.

Požadované dáta získava `Dispatcher` z databázy pomocou dátového pluginu. Inštancia dátového pluginu sa vytvorí pri volaní konštruktora `Dispatchera` a následne pri každom spracovávaní požiadavky. Je to kvôli tomu, že webová aplikácia funguje bezstavovo a preto je nutné všetky nastavenia robiť pri prijatí novej požiadavky znova. Dátový plugin obsahuje funkcie, s ktorými sa s dátovými uzlami a uzlami znalostnej bázy pracuje na úrovni stromov. Tento spôsob práce s dátovou vrstvou bol navrhnutý v [2]. V praxi to vyzerá tak, že existuje abstraktná trieda `DBPlugin`, ktorá len popisuje operácie nad dátovou vrstvou. Pre každý typ databázy (respektíve pre ľubovoľné úložisko dát) sa napíše konkrétna implementácia tejto triedy. Názov implementujúcej triedy je uvedený v konfiguračnom súbore `Dispatchera`. V prípade, že sa správca servera rozhodne, že sa dátová vrstva premiestni z ORACLE na MS SQL Server, tak sa naimplementuje `DBPlugin` pre tento typ databázy a v konfiguračnom súbore sa prepíše len meno triedy. Po reštarte komponentu sa vytvorí inštancia uvedenej triedy a systém pracuje s inou databázou.

`Dispatcher` pri svojej práci využíva okrem dátového pluginu aj služby ďalších komponentov systému – `UserManager` a komponentu `PatientAdminInfo`. Adresy, na ktorých služby bežia, sú uvedené v konfiguračnom súbore, takže v prípade ich presunutia znova stačí editovať len tieto hodnoty.

3.4 Komponent PatientAdminInfo

Táto časť systému je úplne nová a v predošlých implementáciach sa nevyskytovala. Medzi služby, ktoré poskytuje, patrí:

- založenie nového pacienta
- pridanie kontaktu pacientovi
- získanie administratívnych údajov konkrétneho pacienta
- získanie zoznamu všetkých pacientov s ich základnými identifikačnými údajmi
- pridanie zdravotnej poisťovne pacientovi
- zmena atribútu pacienta
- zmena atribútu kontaktu.

Pri zakladaní nového pacienta dostane komponent `PatientAdminInfo` meno, priezvisko a rodné číslo nového pacienta. Z kombinácie týchto hodnôt vygeneruje unikátny identifikátor pacienta a v LDAP strome založí záznam s atribútom UID rovným tejto hodnote. V dátovej vrstve je následne vytvorený koreň nového dátového stromu, ktorý má takisto priradený uvedený identifikátor. Potom sa pridajú pod koreň uzly reprezentujúce meno, priezvisko a rodné číslo (pre spomínanú kompatibilitu so staršími klientmi).

Pridanie kontaktu pacientovi prebieha podobne. Klient poskytne atribúty kontaktu (typ, ulicu, mesto, krajinu, email, telefón) a ID pacienta. V strome v adresárovom serveri aj v dátovom strome v dátovej vrstve sa pridajú podstromy reprezentujúce vytvorený kontakt. Z adresárového servera sa ako odpoveď na tento príkaz vráti DN kontaktu.

Získanie administratívnych údajov pacienta je zrejme najviac využívaná služba komponentu `PatientAdminInfo`. Hľadať sa môže buď podľa trojice meno, priezvisko a rodné číslo, alebo podľa identifikátora pacienta, ak ho má klient k dispozícii. V prípade hľadania podľa identifikátora je hľadanie rýchle – vykoná sa presne jedna operácia, ktorou sa získajú údaje o pacientovi, pretože daný pacient je v LDAP strome identifikovaný svojím ID jednoznačne. Ak sa však vyhľadáva podľa mena, priezviska a rodného čísla, situácia sa mierne komplikuje. Nemôžme totiž nútiť ľudí aby si nemenili mená (hlavne dámy), prípadne rodné čísla. V prípade zmeny niektorého z uvedených atribútov by sa nedalo určiť pôvodné ID (ak by sa pacient napríklad nepamätal na to, ako sa volal pri prvej návšteve, keď bol do systému zavedený, tak by identifikátor vypočítaný z aktuálnych hodnôt bol iný ako ten, ktorým je identifikovaný v LDAP strome) a tým by bolo hľadanie podľa ID neúspešné. Preto sa v tomto prípade postupuje tak, že v podstrome s koreňom, ktorý má DN rovné hodnote

ou=patients,o=patients, sa prechádzajú všetky objekty typu patient a pre každý sa kontroluje atribút birthnum. V prípade zhody sa ešte kontrolujú atribúty sn (pre priezvisko) a givenname (pre krstné meno). Pri zhode všetkých troch atribútov získame záznam, ktorý zodpovedá danému pacientovi. Na obrázku 4 je časť stromu, ktorý je použitý v implementácii komponentu PatientAdminInfo. Sú nakreslené len najdôležitejšie atribúty objektov, všetky sú k dispozícii v schéme LDAP stromu v prílohe 1.

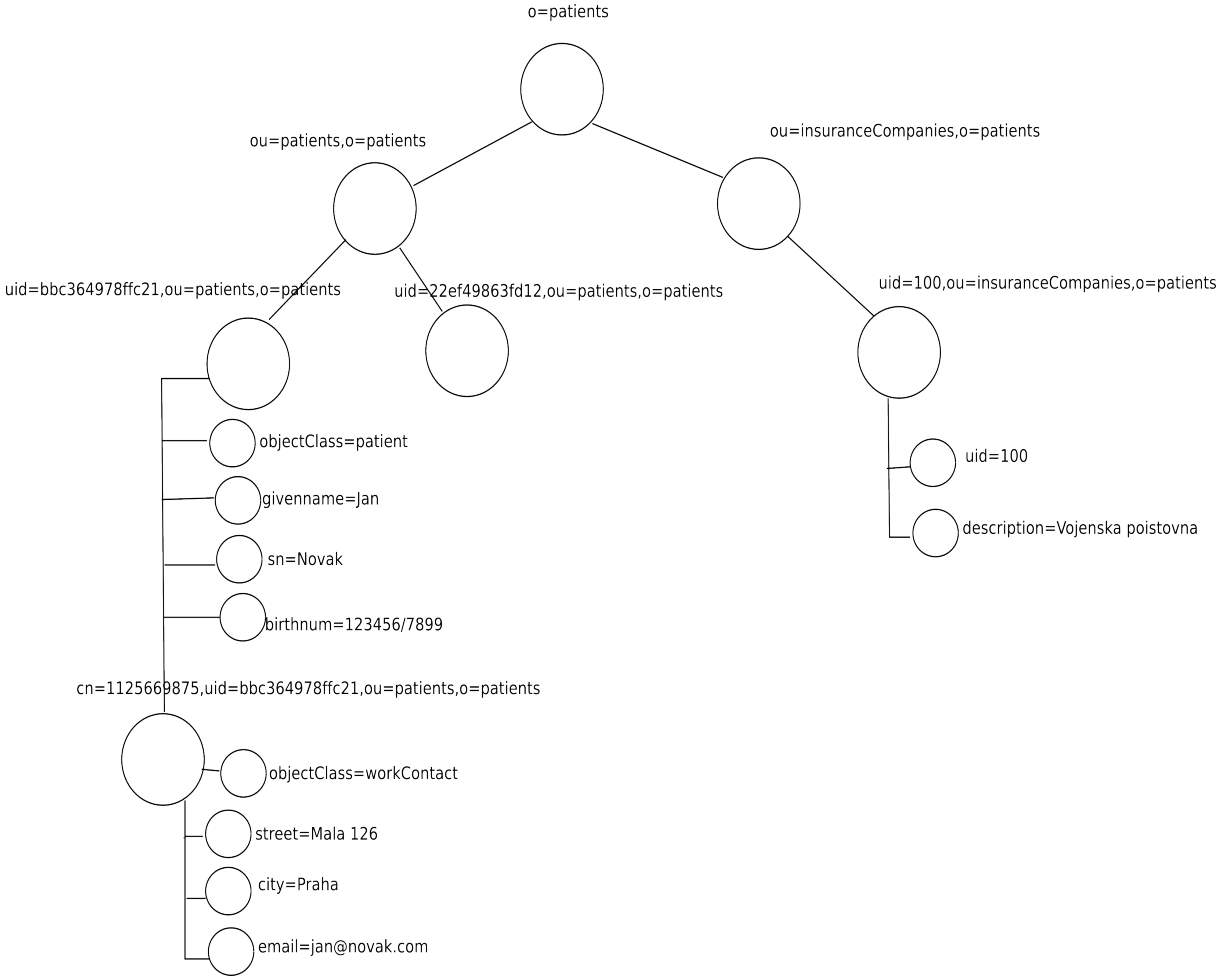
Na obrázku 4 je možné si všimnúť spôsob vytvárania DN jednotlivých objektov. Atribút a hodnota, ktoré sú uvedené len samotne (t.j. bez čiarky v názve), tvoria RDN[3] záznamu. Ak chceme získať DN záznamu, musíme vziať jeho RDN, vystúpiť o úroveň vyššie, pripojiť RDN otca a tento postup opakovať až kým nedorazíme ku koreňu.

RDN musí tvoriť aspoň jeden atribút s hodnotou. V prípade, že ho tvorí viacero atribútov, tak sú oddelené znakom „+“. Na obrázku 4 vidíme, že koreňom je uzol s DN aj RDN o=patients. Ten má dve deti, jedno z nich má RDN ou=patients, druhé ou=insuranceCompanies. Prvý uzol združuje záznamy o pacientoch, druhý zase zoznam poisťovní. Oba tieto uzly sú typu structural, čo im umožňuje mať potomkov.

Ak zostúpime o úroveň nižšie v podstrome s koreňom s RDN ou=patients, dostali sme sa na zoznam pacientov. Každý má svoje RDN, ktoré má tvar „uid=hodnota“. Objekty sú typu (atribút objectClass) patient a tento typ dedí od typu structural, čo umožňuje pacientom „mať potomkov“. Potomkami sa rozumejú kontakty, ktorých RDN je číselná hodnota (systémový čas pri vytvorení). Aj v rámci LDAP objektov existuje možnosť „virtuality“ a teda je zadefinovaný abstraktný typ Contact, od ktorého dedia typy homeContact, workContact, postalContact a otherContact, ktoré sa môžu líšiť svojimi atribútmi (musia však zachovávať atribúty predka).

Jednotlivé atribúty kontaktu sa v jednom kontakte vyskytujú maximálne raz. Kontakt sa chápe ako nejaká „vizitka“ pacienta. Ak sa uvádza viacero adries alebo telefónnych čísel, chápe sa to ako viac rôznych kontaktov.

3 Relative Distinguished Name – slúži na jednoznačnú identifikáciu uzlu LDAP stromu medzi jeho súrodencami



Obrázok 4: Časť LDAP stromu používaného v komponente PatientAdminInfo

3.4.1 Implementácia adresárového servera

V súčasnej dobe je k dispozícii niekoľko adresárových serverov, komerčných (Active Directory od Microsoftu), aj open sourcových (Apache Directory Server, OpenDS a.i.). Vzhľadom k použitiu aplikačného servera Tomcat a nástroja Axis2, ktoré pochádzajú od Apache Software Foundation, padla moja voľba na Apache Directory Server (ApacheDS 1.0 [9]). Toto riešenie je open-source a takisto pochádza z dielne spoločnosti Apache ako aj Tomcat a Axis2. Server je napísaný v jazyku Java a je certifikovaný ako LDAP v3 [8] server. Priamo pre tento server existuje aj klient Apache Directory Studio (predtým LDAP Studio), ktorý bol pre svoje prívetivé užívateľské prostredie použitý na tvorbu a testovanie LDAP schémy.

V ApacheDS sa záznamy ukladajú v tzv. partíciach. Každá partícia obsahuje kompletný strom záznamov, ktorý sa tiež niekedy nazýva DIT (Directory Information Tree). Záznamy v rozličných partíciach sú na sebe nezávislé. Záznamy v jednej partícii majú spoločný znak - nazýva sa „partition suffix“ a je to koncovka, ktorú obsahuje DN každého záznamu v tejto partícii. Pridávanie nových partícií sa dá vykonať editovaním konfiguračného súboru LDAP servera (`conf/server.xml` v inštalačnom adresári ApacheDS). Podrobný návod je k dispozícii v [14].

Pre účely ukladania administratívnych údajov pacientov bola vytvorená samostatná partícia, ktorej partition suffix je „o=patients“. V konfiguračnom súbore jej nastavenie vyzerá nasledovne:

```
<bean id="patientsPartitionConfiguration"
class="org.apache.directory.server.core.partition.impl.btree.Mutable
BTreePartitionConfiguration">
    <property name="name" value="pacienti" />
    <property name="cacheSize" value="100"/>
    <property name="suffix" value="o=patients" />
    <property name="optimizerEnabled" value="true" />
    <property name="synchOnWrite" value="true" />
    <property name="contextEntry">
       <value>
         objectClass: top
         objectClass: organization
         objectClass: extensibleObject
         o: patients
       </value>
    </property>
 </bean>
```

Meno partície je „pacienti“, ako suffix sa používa hodnota „o=patients“. Element `contextEntry` hovorí, že typ tohto objektu je „top“, „organization“ a „extensibleObject“ a teda slúži ako koreň DIT. Koreňu je potrebné ešte pridať dvoch potomkov – uzly s RDN `ou=patients` a `ou=insurnaceCompanies`. Tie sa dajú do adresárového servera vložiť pomocou príkazového riadku z LDIF súboru. LDIF (LDAP Data

Interchange Format) je štandard definovaný v RFC2849 [10] a slúži na výmenu LDAP záznamov medzi LDAP servermi.

Súbor `patients.ldif` je na priloženom CD pri inštalačnom súbore ApacheDS. Dáta uvedené v tomto súbore sa vkladajú príkazom

```
>java -jar apacheds-tools.jar import -e -f patients.ldif -h
host -p port -u user -w adminPassword
```

Archív `apacheds-tools.jar` sa nachádza v adresári `bin` inštalačného adresára ApacheDS. Ako host sa uvedie meno stroja (prednastavená hodnota je localhost), kde je LDAP server spustený, užívateľ je prednastavený na „uid=admin,ou=system“ a heslo na „secret“. Tieto hodnoty sú nastavené pri inštalácii ApacheDS a odporúča sa ich zmeniť.

Po vytvorení partície a základnej štruktúry na ukladanie administratívnych údajov pacientov je potrebné ešte popísať, aké atribúty budú ukladané záznamy obsahovať. Tento popis je uvedený v LDAP schéme. Schéma obsahuje popis objektov, ktoré sa v strome smú vyskytovať a zoznam ich povinných a nepovinných atribútov. Takisto obsahuje definície používaných atribútov. Triedy objektov od seba môžu dediť – presne ako v objektovo orientovanom programovaní. V prílohe 1 je uvedená kompletná schéma, ktorá bola použitá v tejto práci.

ApacheDS 1.0 zatiaľ neposkytuje možnosť dynamickej zmeny schémy, preto je nutné najprv vytvorenú schému skompilovať, umiestniť do adresára `lib` v inštalačnom adresári ApacheDS, uviesť meno schémy do konfiguračného súboru servera a server reštartovať. Po reštarte je možné vytvárať inštancie objektov definovaných v schéme. Skompilovaná schéma `patients-schema.jar` je priložená na CD. Celý postup, ako sa schéma kompiluje, je k dispozícii na adrese

`http://directory.apache.org/apacheds/1.0/custom-schema.html`

a preto tu nebude podrobne popisovaný.

Inou možnosťou implementácie oddeleného skladu administratívnych údajov bolo použitie konceptu PIDS (Person Identification Service) [16]. Uvedený spôsob je založený na použití middlewaru CORBA a je navrhnutý pre všeobecné ukladanie a získavanie identifikačných údajov osôb. Pretože v predkladanej práci sa na prenos správ medzi komponentmi využíva princíp web services (nástroj Axis2) a nie CORBA, bolo rozhodnuté použiť klasický adresárový server (LDAP) a vytvoriť vlastnú schému na ukladanie administratívnych údajov pacientov.

3.5 Komponent UserManager

Tento komponent má na starosti prácu s užívateľmi servera elektronického zdravotného záznamu. Je implementovaný ako webová služba, ktorá poskytuje nasledujúce operácie:

- autentifikácia užívateľa
- pridanie nového užívateľa
- získanie zoznamu užívateľov
- získanie informácií o konkrétnom užívateľovi
- zmena atribútov užívateľa (meno, priezvisko, heslo, práva).

Autentifikácia užívateľa prebieha vždy po prijatí požiadavky. Jej výsledkom je odpoveď true/false. Ak užívateľ nie je autentifikovaný, server zvyšok požiadavky odmietne a vráti hlásenie o chybe.

Pre pridanie nového užívateľa je potrebné uviesť jeho meno, priezvisko, typ a heslo. Po úspešnom pridaní sa v odpovedi vráti ID novo pridaného užívateľa. Ten potom môže serveru posielať požiadavky, v ktorých sa bude identifikovať buď svojím ID alebo menom a priezviskom, ktoré uviedol pri zakladaní svojho účtu.

V komunikačnom protokole je definovaných 7 typov užívateľov – administrátor, programátor, sestra, výskumník, študent, lekár a „iný“. Každá skupina má rôzne oprávnenia, ktoré sa týkajú buď modifikácie znalostnej bázy alebo prehliadania a editovania dátových stromov. Problematika toho, kto má aké práva k jednotlivým prostriedkom, je riešená v [1].

3.6 Komponent Data Warehouse

Táto časť systému má na starosti nahrávanie multimediálnych súborov do databázy. Podobne ako `Dispatcher`, aj ona využíva databázový plugin pre pripojenie k dátovej vrstve a tým je nezávislá na použitej databáze.

Je umiestená v kontajneri aplikačného serveru, kde prijíma požiadavky od klientov. Tí jej požiadavky zasielajú metódou HTTP POST. V každej požiadavke sa musia vyskytnúť atribúty *userName* a *userPassword*, ktoré obsahujú identifikačné údaje užívateľa. Nasledujú atribúty *resource* a *object*. Prvý z nich obsahuje číslo dátového uzlu, ku ktorému multimediálne dáta patria. Druhý obsahuje MIME[4] typ vkladaného objektu. Všetky tieto atribúty musia byť uvedené. Užívateľ je najprv autentifikovaný pomocou komponentu `UserManager` a až potom sa vykoná samotné nahrávanie dát.

Po štyroch spomenutých atribútoch, ktoré sú v tvare „atribút=hodnota“, nasledujú 2 prázdne riadky a po nich multimediálne dáta v kódovaní

4 http://www.faqs.org/rfcs/rfc2045.html

Base64. Dátový sklad tieto dáta rozkóduje späť do pôvodnej podoby (pole bajtov) a uloží ich s ostatnými informáciami do dátovej vrstvy.

Maximálna veľkosť vkladaného súboru je obmedzená niekoľkými faktormi. Prvým je maximálna veľkosť typu Integer. Je to preto, lebo pole bytov, na ktoré je Base64 reťazec dekódovaný, obsahuje prijaté dáta v celku. Maximálna veľkosť tohto poľa závisí od maximálnej hodnoty typu Integer. V Jave je toto obmedzenie zhruba 2 GB. Tieto 2GB platia však pre Base64 reťazec, nie pre skutočné dáta. Ak sa dáta zakódujú do Base64, trojica bytov sa reprezentuje štvoricou bytov, čo znamená zväčšenie objemu zhruba o tretinu. V skutočnosti je teda možné nahrať súbor o veľkosti približne 1,7 GB. Ďalšími obmedzeniami môžu byť používaný súborový systém na pevnom disku a maximálna veľkosť haldy pre virtuálny stroj jazyka Java (ktorá však môže byť zmenená pri spúšťaní virtuálneho stroja prepínačom -Xmx).

Nakoľko server slúži na uchovanie rôznych čiernobielych snímkov (RTG...), prípadne krátkych audiovizuálnych záznamov veľkosti rádovo MB, je toto riešenie v súčasnosti postačujúce.

Sťahovanie multimediálnych súborov zabezpečuje priamo `Dispatcher` prostredníctvom funkcie `get_data_file`. Dáta z dátovej vrstvy zakóduje do Base64 a odošle klientovi v podobe hodnoty požadovaného uzla.

3.7 Klient EHRclientJ

Pre testovanie počas vývoja bol použitý klient EHRc, ktorý bol súčasťou riešenia v [1]. Tento jednoduchý klient na vstupe vzal XML súbor, ktorý odoslal serveru a zobrazil odpoveď. Jeho nedostatkom je, že nevie korektne spracovávať kódovanie znakov v XML a napevno pracuje s kódovaním windows-1250.

Preto bol v jazyku Java naprogramovaný podobný klient (s názvom EHRclientJ), ktorý vstupný dokument najprv načíta v korektnom kódovaní t.j. takom, ktoré je uvedené v hlavičke XML dokumentu a potom pošle túto požiadavku serveru. Server vo svojich odpovediach používa kódovanie UTF-8.

Tento klient takisto poskytuje možnosť posielať multimediálne súbory dátovému skladu. Na príkazovom riadku sa uvedie súbor na disku, MIME typ jeho obsahu a identifikačné číslo (alebo užívateľské meno) a heslo užívateľa. Klient vezme obsah tohto súboru, zakóduje ho do Base64 a metódou HTTP POST ho spolu s potrebnými atribútmi odošle dátovému skladu.

Ďalším nedostatkom starého testovacieho klienta bolo, že jeho nastavenie (server, ktorému posiela požiadavky) bolo uložené v registroch. EHRclientJ používa konfiguračný súbor `client.properties`, v ktorom pri zmene adresy servera stačí zmeniť hodnotu atribútu `server.location`, prípadne `dwh.location` pre dátový sklad.

Klient uloží odpoveď od servera do súboru `response.tmp.xml` vo svojom pracovnom adresári.

3.8 Bezpečnosť a možnosť šifrovania dát

Jednou z požiadavok kladených na túto prácu bola možnosť šifrovania komunikácie medzi klientom a serverom a možnosť digitálne podpisovať vybrané dáta.

3.8.1 Šifrovanie prenosu dát medzi serverom a klientom

Na túto úlohu je vhodné použiť ako prenosový protokol medzi klientom a serverom protokol SSL, ktorému sa venuje nasledujúci text.

Táto práca nemá za úlohu podrobne skúmať problematiku šifrovania dát, je však vhodné uviesť základné pojmy a princípy ich fungovania.

Certifikát je dokument vydaný certifikačnou autoritou (CA) a má za úlohu potvrdiť fakt, že jeho držiteľ je skutočne ten, za koho sa vydáva. Skutočný a platný certifikát vydaný certifikačnou autoritou (napríklad VeriSign, Thawte…) je dosť drahý a preto sa certifikátmi preukazujú väčšinou len servery, od klientov nie sú vyžadované (aj keď je v rámci SSL možné vyžadovať certifikát aj od klienta). Pre testovacie účely sa certifikát dá vygenerovať na vlastnom počítači.

SSL[5] je skratka pre Secure Sockets Layer a tento protokol bol vyvinutý firmou Netscape pre prenos privátnych dokumentov cez internet, ktorý sa považuje za nebezpečný, pretože komunikácia môže byť pomerne ľahko odpočúvaná. Protokol bol prijatý ako štandard Internet Engineering Task Force (IETF). SSL poskytuje možnosť autentifikácie servera voči klientovi (a naopak) a možnosť šifrovania komunikácie medzi serverom a klientom. Ak chce klient nadviazať SSL spojenie so serverom, začína proces tzv. handshakingu, počas ktorého sa vykonávajú tieto kroky:

(1) klient pošle serveru požiadavok na komunikáciu cez SSL

(2) server odpovie poskytnutím svojho certifikátu a prípadným vyžiadaním certifikátu od klienta

5 http://www3.ietf.org/proceedings/95apr/sec/cat.elgamal.slides.html

(3) klient validuje certifikát servera – autentizuje ho voči známym certifikačným autoritám (v prípade, že je CA neznáma, klient sa opýta užívateľa, či certifikát prijať a zobrazí jeho vydavateľa, dátum platnosti a podpis vydavateľa); klient ešte skontroluje tzv. Certification Revocation List – zoznam neplatných certifikátov

(4) v prípade, že je certifikát platný, klient vytvorí Master secret key, ktorý zašifruje použitím verejného kľúča servera (ten získa z certifikátu) a odošle ho serveru; v prípade, že to server vyžadoval, odošle aj svoj certifikát, ktorý je serverom validovaný rovnakým spôsobom ako na strane klienta

(5) klient aj server použijú Master secret key na vygenerovanie symetrického kľúča, ktorý je používaný k šifrovaniu a dešifrovaniu prenášaných správ počas prebiehajúcej relácie.

Počas handshakingu sa vyjednávajú aj niektoré ďalšie atribúty spojenia, napríklad používané šifrovacie algoritmy.

Komunikácia cez SSL je plne v réžii aplikačného servera, na ktorom je server elektronického zdravotného záznamu spustený. V našom prípade sa jedná o aplikačný server Tomcat 6.0. Postup pre umožnenie SSL komunikácie je dostupný v [12] a budeme sa mu venovať v kapitole 5.

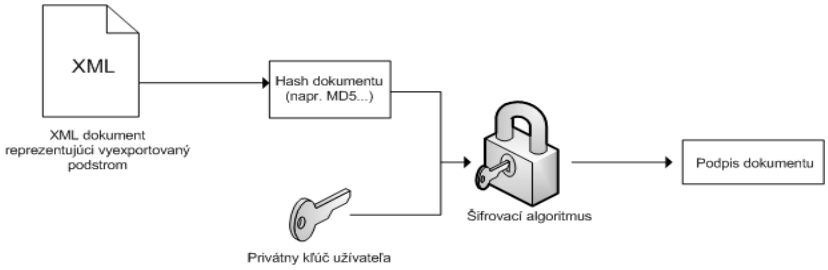
3.8.2 Podpisovanie údajov

V mnohých profesiách a najmä v medicíne je potrebné nejakým spôsobom podpisovať dáta, čo znamená zabezpečiť ich autenticitu a integritu (nemožnosť zmeny treťou stranou). Tento problém riešia digitálne podpisy.

Predstavme si, že lekár určí pacientovi liečbu na základe hodnôt, ktoré nameral pri vyšetrení a uložil v dátovom strome pacienta. Jediným potvrdením, že tam tieto hodnoty vložil daný lekár, je jeho ID uložené spolu s dátami. Nič ale nebráni zmene priamo v tabuľkách dátovej vrstvy a teda nie je zaistená integrita dát. Preto je niekedy ešte potrebné, aby boli dáta tzv. digitálne podpísané a spolu s podpisom uložené na miesto na to určené.

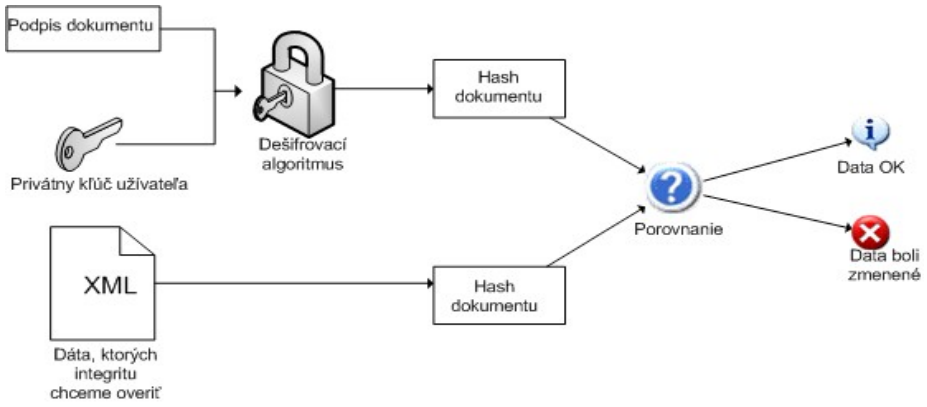
Užívateľ systému preto musí vlastniť privátny kľúč, ktorý pozná len on sám. Pomocou funkcie `get_data_to_sign` si vypýta podstrom z dátového stromu pacienta, ktorý chce potvrdiť svojim podpisom. Server mu ako odpoveď na túto požiadavku vráti požadovaný podstrom vyexportovaný vo formáte XML. Užívateľ pomocou svojho privátneho kľúča vykoná samotný akt podpisu – v praxi to môže vyzerať tak, že sa zráta hash[6] zo získaného XML, na ktorý sa aplikuje šifrovací algoritmus za pomoci privátneho kľúča. Tento podpis je odoslaný na server funkciou `enter_signed_data`.

6 Výsledok kryptografickej hashovacej funkcie (MD5, SHA-1...)



Obrázok 5: Schéma procesu podpisovania údajov

V tabuľke EHR_SIGNED_DATA sa tak vedľa seba ocitnú pôvodne vyexportované dáta a ich podpis. Pri zisťovaní integrity pôvodných dát sa jednoducho zoberú dáta, ktoré chceme overiť a zrátame ich podpis presne tak, ako to robil klient pri podpisovaní (na vypočítanie podpisu je nutný privátny kľúč dotyčného užívateľa, tým pádom jeho spolupráca). Ak sa výsledok tejto operácie rovná podpisu, ktorý je uložený v databáze, môžeme tvrdiť, že dáta nikto nezmenil a že boli skutočne zapísané daným užívateľom.



Obrázok 6: Schéma procesu overovania integrity dát

Popísaný spôsob overovania integrity narazí na problém, ak sa zo strany užívateľa vyskytne neochota spolupracovať na overovaní. Preto by stálo za uváženie použiť dvojicu súkromný a verejný kľúč. Verejný kľúč každého užívateľa by bol prístupný verejne a pri overovaní integrity stačí podpis dešifrovať verejným kľúčom a získaný hash porovnať s hashom overovaných dát. Táto metóda (tiež známa ako asymetrické šifrovanie alebo public-key cryptography [13]) sa v praxi používa častejšie.

Kapitola 4
Programátorská dokumentácia

Táto kapitola sa bude venovať popisu fungovania servera elektronického zdravotného záznamu a jeho komponentov z pohľadu programátora. Zdrojový kód bol počas vývoja v značnej miere komentovaný. Tieto komentáre boli vyexportované nástrojom Javadoc a sú k dispozícii na priloženom CD.

V dnešnej dobe je často kladená požiadavka na platformovú nezávislosť software. Táto nezávislosť sa dá dosiahnuť buď pomocou napísania samostatnej verzie pre každú platformu, alebo podsunutím virtuálneho stroja pod bežiacu aplikáciu.

Možnosť implementácie v jazyku C/C++ bola zavrhnutá hneď na začiatku, pretože zabezpečiť platformovú nezávislosť v tomto prípade je dosť obtiažne. Okrem toho, programy písané v týchto jazykoch obsahujú veľké množstvo tzv. zavlečenej zložitosti – tú tvoria technické problémy, ktoré musí programátor riešiť navyše oproti pôvodnému problému (napr. memory management).

Ďalšou možnosťou bolo použiť jazyk C# a platformu .NET. Tým by sme získali program, ktorý by bolo možné spúšťať na ľubovoľnom PC s nainštalovaným .NET frameworkom. Bohužiaľ, .NET framework zatiaľ existuje len pre platformu Win32 a čiastočne pre Linux (open-source projekt Mono).

Ako implementačný jazyk bol preto zvolený jazyk Java, konkrétne verzia Java 2 Standard Edition 5.0 (bežne označovaná ako Java 1.5) . Táto verzia je oproti Jave 1.4 vylepšená o generické typy, automatické obaľovanie primitívnych typov a iné vlastnosti. Užívateľ potrebuje pre spustenie programu Java Runtime Enviroment, ktoré sa dá zdarma stiahnuť na stránkach spoločnosti SUN a nachádza sa aj na priloženom CD (JRE 1.5_06 pre platformu Windows-32bit).

Komponenty `UserManager` a `PatientAdminInfo` sú exportované ako webové služby. Sú zabalené vo webových archívoch, ktorých adresárová štruktúra je popísaná v úvode kapitoly 3.

Komponenty `Dispatcher` a `Data Warehouse` sú exportované ako aplikácie pre aplikačný server Tomcat (servlety). Ich štruktúra je podobná ako pri webových archívoch a vyzerá nasledovne:

```
-application.war
   - *.html  a *.jsp súbory
   - WEB-INF
      -classes
         -myPackage
            myClass.class
      -lib
      -web.xml
```

V adresári lib sú umiestnené knižnice, ktoré aplikácia využíva. Súbor `web.xml` obsahuje informácie o aplikácii. Najdôležitejšou informáciou je plné meno triedy, ktorá rozširuje abstraktnú triedu `HttpServlet`. Po prijatí požiadavky sa volá metóda `service` tejto triedy, ktorá potom podľa druhu požiadavku zavolá ďalšie metódy (najčastejšie metódy `doGet` a `doPost`). Aplikácia môže obsahovať viacero servletov, a preto je každý z nich zadefinovaný v samostatnom elemente `servlet`.

V súbore sú takisto zadefinované mapovania servletov na URL adresy. V každom elemente `servlet-mapping` sa uvedie meno servletu a URL adresa, na ktorej prijíma a odosiela požiadavky. Pre názornosť je uvedený príklad:

```
<servlet>
      <servlet-name>dispatcher</servlet-name>
      <description>
        This servlet accepts requests from clients and returns
       responses to them.
       </description>
      <servlet-class>mudrj.dispatcher.Dispatcher</servlet-class>
      <load-on-startup>5</load-on-startup>
</servlet>
```

Tento element zo súboru `web.xml` definuje servlet s menom `dispatcher` a trieda, ktorá rozširuje `HttpServlet` je `mudrj.dispatcher.Dispatcher`. Tento servlet začne prijímať požiadavky po štarte aplikačného servera.

```
<servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/dispatcher</url-pattern>
</servlet-mapping>
```

Týmto elementom je povedané, že servlet `dispatcher` je k dispozícii na adrese, ktorá vznikne pripojením reťazca „/dispatcher“ k tzv. context-path. Každá aplikácia, ktorá beží na aplikačnom serveri, má priradený reťazcový identifikátor nazývaný context-path, podľa ktorého je na serveri rozlíšiteľná.

V našom prípade má aplikácia priradený context-path `mudrj`. Teda `Dispatcher` bude prístupný na adrese

`host:port/mudrj/dispatcher`.

4.1 Spracovanie požiadavky klienta

Z doterajšieho textu je zrejmé, že vstupným bodom servera je trieda `Dispatcher`. Tá rozširuje `HttpServlet` a preťažuje jeho metódu `doPost`. Klient odošle svoju požiadavku vo forme XML súboru HTTP metódou POST, do ktorej tela vloží obsah daného súboru. V metóde `doPost` na serveri sa inicializuje `DBPlugin`, načíta sa súbor s nastaveniami a pripojí sa k dátovej vrstve. Následne sa prečíta vstup a očistí od všetkých znakov pred úvodným znakom '<' a za posledným znakom '>'. Takto získané XML sa predá funkcii `processCommand`. Táto funkcia vytvorí novú inštanciu triedy `InputRequest` a ako parametre jej predá nainicializovaný DBPlugin, inštanciu triedy `Properties` s načítaným konfiguračným súborom a vstupné XML. Na vytvorenej inštancii sa volá funkcia `executeCommands`, ktorej výstupom je `Document` reprezentujúci odpoveď servera.

Trieda `InputRequest` počas svojho vytvárania zvaliduje vstupné XML podľa schémy uvedenej v konfiguračnom súbore, vytvorí prázdny výstupný dokument a vytvorí inštancie objektov reprezentujúcich vzdialený `UserManager` a `PatientAdminInfo`. Pri vykonávaní metódy `executeCommands` sa najprv autentifikuje užívateľ, ktorý zaslal požiadavku. Po úspešnej autentifikácii sa spracuje zvyšok požiadavky a to tak, že sa vytvorí pre každý element `command` inštancia triedy `MudrCommand` a uloží sa do fronty určenej na spracovanie. Postupne sa vykonávajú všetky príkazy vo fronte a svoje odpovede v elementoch `response` vkladajú do výstupného dokumentu. Ten je po vykonaní všetkých príkazov ešte zvalidovaný rovnako ako na vstupe. Kvôli kompatibilite s MUDRc-om je ešte nutné zmeniť vo výstupe dátové typy „LDAP referencia“ na „directory“ a typ „dátum“ na boolean“ (tu však navyše treba nastaviť údaj platnosti na hodnotu, ktorá bola uvedená pôvodne).

Dokument je následne vrátený o úroveň vyššie, prevedený na String a odoslaný naspäť klientovi.

4.2 Vykonávanie príkazov

Vykonávanie príkazov má na starosti trieda `MudrCommand`. Jej inštancie vytvára trieda `InputRequest` pri parsovaní vstupného XML. V konštruktore dostane všetky objekty, ktoré by mohla potrebovať

k vykonaniu príkazu. Má jedinú verejnú metódu menom `executeCommand`, ktorá robí to, že vezme samotný obsah príkazu a spracuje ho. To znamená, že sa pozrie na meno elementu a na základe toho zavolá jednu z privátnych obslužných metód `process"MenoPríkazu"`. Každá z týchto obslužných metód zistí potrebné parametre príkazu, ktorý má vykonať, vykoná ho a svoju odpoveď zapíše v určitom tvare (podľa schémy z konfiguračného súboru) do elementu `response`. Tento element je potom vložený ako dieťa koreňového elementu výstupného dokumentu.

4.3 PatientAdminInfo a UserManager

Webová služba UserManager používa ako svoju service-class[7] triedu `UserManagerService`. Trieda obsahuje privátnu položku typu `IUserManager`, kde `IUserManager` je rozhranie, ktoré popisuje, aké metódy musí UserManager podporovať. Metódy sú implementované v našom prípade v triede `UserManager`, ktorá informácie o užívateľoch získava z databázy Oracle. Pre zmenu implementačnej triedy stačí prepísať hodnotu atribútu `mudrjUM.impl` v súbore s nastaveniami komponentu UserManager a samozrejme dodať triedu, ktorá implementuje všetky funkcie rozhrania. Databázová nezávislosť je v tomto prípade zabezpečená výmenou implementačnej triedy rozhrania `IUserManager`, ktorá plní funkcie databázového pluginu.

Služba PatientAdminInfo poskytuje administratívne údaje o pacientoch, ktoré sa zhromažďujú na LDAP serveri. Tieto údaje sa dajú vyžiadať prostredníctvom triedy `Main`, ktorá obsahuje metódy na získavanie pacientov a kontaktov. Pri volaní metód sa vytvorí inštancia `EntryHandleru` a to buď `PatientHandler` alebo `ContactHandler`. Táto inštancia sa spojí s LDAP serverom a získava údaje pomocou tried a metód z knižnice `ldap.jar`. Výsledok je zabalený do objektu z balíčka `mudrj.patientAdmin.model` a odoslaný späť klientovi.

4.4 Triedy RemotePatientAdmin a RemoteUserManager

Tieto triedy v komponente `Dispatcher` plnia úlohu klientov voči komponentom `PatientAdminInfo` a `UserManager`. Zaobaľujú vzdialené metódy a vytvárajú dojem „lokálnosti“. To znamená, že Dispatcher si vytvorí inštancie týchto tried, ktorým nastaví URL adresy, na ktorých bežia spomenuté komponenty ako webové služby a všetky operácie, ktoré by vykonával na vzdialených strojoch, vykonáva lokálne

7 Viď kapitolu 3.1

prostredníctvom uvedených `Remote*` tried. V prípade naimplementovania modulu lekárskych doporučení je jeho doplnenie jednoduché – stačí napísať triedu, ktorá bude obaľovať vzdialené metódy tohoto modulu.

4.5 Ostatné triedy a balíky

Každý komponent obsahuje balíček `utils.*`. V tomto balíčku sú rôzne pomocné triedy a funkcie, ktoré uľahčujú prácu s XML dokumentom, prácu s typom Date, kodéry a dekodéry Base64 a triedy, ktoré uľahčujú prácu s dátovým pluginom. Tieto triedy reprezentujú riadky niektorých tabuliek v databáze. Napríklad trieda `DomainInfo` obsahuje číslo domény a id jej koreňa (tabuľka KNOWLEDGE_DOMAINS má stĺpce `id` a `kn_id`). Po vykonaní príkazu dátového pluginu `getKnowledgeDomains` sa výsledný `ResultSet` zabalí do triedy `SqlResultDomainInfo`. Tento objekt potom pri volaní metódy `next()` vracia inštancie triedy `DomainInfo`.

4.6 Vkladanie multimediálnych dát

Funkcionalitu pre vkladanie multimediálnych dát poskytuje komponent `Data Warehouse`, ktorý je podobne ako `Dispatcher`, exportovaný ako servlet. Požiadavky dostáva metódou HTTP POST. Podrobnosti týkajúce sa vkladania multimédií sú pri popise implementácie komponentu `Data Warehouse` v kapitole 3.

Kapitola 5
Inštalácia servera

Inštalácia samotných komponentov pozostáva len zo skopírovania archívov z adresára `dist` na priloženom CD do adresárov, ktoré budú uvedené v ďalšom texte. Je možné mať každý komponent spustený na samostatnom počítači, ale potom je nutné každý takýto počítač vybaviť požadovaným software.

K úspešnému spusteniu servera elektronického zdravotného záznamu je potrebný tento software:

1. Java 2 Runtime Enviroment 5.0 – verzia pre platformu Windows je priložená na CD. Počas inštalácie by sa mala automaticky nastaviť premenná JAVA_HOME. Java Runtime Enviroment je potrebné pre akékoľvek ďalšie kroky inštalácie.
2. Aplikačný server Tomcat 6.0 – bola použitá verzia 6.0.10, ktorej inštalačný súbor je na priloženom CD. Počas inštalácie by sa mala nastaviť premenná CATALINA_HOME. Server je nakonfigurovaný na port 8080, SSL nie je povolené. Pre zmenu je potrebné editovať konfiguračný súbor `conf/server.xml` v inštalačnom adresári Tomcata. Podrobnejšie sa nastaveniu aplikačného servera venuje ďalší text.
3. Axis2. Verzia 1.2 je na priloženom CD ako webový archív. Tento archív stačí umiestniť do adresára s webovými aplikáciami aplikačného servera. Štandardne je to adresár `webapps` v inštalačnom adresári Tomcata (v ďalšom texte je tento inštalačný adresár označovaný názvom CATALINA_HOME). Tomcat 6.0 má vlastnosť Hot Deployment, čo znamená, že po pridaní aplikácie nie je nutné reštartovať server, ale server zistí jej prítomnosť sám, priradí jej context-path a sprístupní ju (tento proces sa nazýva deployment). Po umiestnení archívu `axis2.war` do adresára `CATALINA_HOME/webapps` sa tento archív sám rozbalí do adresára `axis2`, ktorý sa vytvorí takisto v adresári `CATALINA_HOME/webapps`.
4. Adresárový server. V súčasnej implementácii je použitý Apache Directory Server 1.0. Po nainštalovaní je potrebné vytvoriť partíciu pre ukladanie administratívnych údajov pacientov, vytvoriť potrebnú stromovú štruktúru a umožniť vytváranie objektov z vlastnej schémy. Popis týchto akcií sa nachádza pri popise komponentu

`PatientAdminInfo` v kapitole 3 v časti, ktorá sa venuje implmentácii adresárového servera.

Správnosť inštalácie Tomcata si môžeme overiť napríklad v prehliadači zadaním adresy `localhost:8080`. Ak Tomcat funguje, objaví sa uvítacia stránka. Pre overenie inštalácie Axis2 je potrebné v prehliadači zadať adresu `localhost:8080/axis2`. Objaví sa uvítacia stránka Axis-u, na ktorej je potrebné kliknúť na odkaz Validate. Dostaneme sa na stránku, ktorá obsahuje rôzne informácie o systéme, aplikačnom serveri a samotnom Axis-e. Prípadné problémy s prostredím Java alebo Axis2 sa zobrazia červenou farbou.

Ak sú Tomcat aj Axis2 v poriadku, môžeme pristúpiť k inštalácii jednotlivých komponentov. Archívy `PatientAdminService.jar` a `UserManagerService.jar` stačí umiestniť do adresára `CATALINA_HOME/webapps/axis2/WEB-INF/services`. Archívy `mudrj.war` a `dwh.war` je potrebné umiestniť do adresára `CATALINA_HOME/webapps`. Ako už bolo spomenuté, reštart aplikačného servera nie je potrebný.

Každý komponent servera obsahuje adresár `properties`, v ktorom sa nachádza súbor s príponou `.properties`. V tomto súbore sú uvedené nastavenia daného komponentu. Nastavenia majú tvar `parameter=hodnota`. Pred a za symbolom „=“ nesmú byť medzery.

Klienta EHRclientJ nie je potrebné nijak inštalovať, spúšťa sa príkazom

```
>java -jar ehrc.jar fileWithRequest
```

Pre komunikáciu cez SSL je potrebné dodať ešte súbor s certifikátom servera v prípade, že server má „self-signed“ certifikát - to znamená, že nebol vydaný žiadnou dôveryhodnou certifikačnou autoritou a nie je možné ho overiť (s procesom overovania je možné sa stretnúť napríklad pri webových prehliadačoch, keď sa pýtajú, či zobrazený certifikát možno považovať za platný). Klient sa v tomto prípade spúšťa s nastaveniami:

```
>java -Djavax.net.ssl.trustStore="pathToKeystore"
      -Djavax.net.ssl.trustStorePassword="KeystorePassword"
      -jar ehrc.jar fileWithRequest
```

V procese SSL handshakingu po príjme certifikátu od servera sa prijatý certifikát hľadá medzi certifikátmi v uvedenom súbore (javax.net.ssl.trustStore). Pre potreby SSL komunikácie s dátovým skladom je nutné do tohto súboru naimportovať aj certifikát servera, na ktorom je komponent `Data Warehouse` spustený.

Pre vkladanie multimediálnych dát sa klient spúšťa volaním

```
>java -jar ehrc.jar -mm multimediaFile -mime mime-type -id
data_file_id -userid iduzivatela -userpwd heslouzivatela
```

5.1 Konfigurácia aplikačného servera Tomcat 6.0 pre SSL komunikáciu

Pre povolenie SSL spojenia s aplikačným serverom je potrebné v konfiguračnom súbore definovať tzv. connector pre ľubovoľný voľný port (SSL používa štandardný port 443, Tomcat štandardne používa pre SSL spojenia port 8443). Do súboru `conf/server.xml` v inštalačnom adresári Tomcata stačí dopísať tento connector:

```
<Connector
          protocol="org.apache.coyote.http11.Http11Protocol"
          port="8443" minSpareThreads="5"
          maxSpareThreads="75"
          enableLookups="true" disableUploadTimeout="true"
          acceptCount="100"  maxThreads="200"
          scheme="https" secure="true" SSLEnabled="true"
          keystoreFile=".keystore" keystorePass="euromise"
          clientAuth="false" sslProtocol="SSL"/>
```

Dôležité nastavenia sú `keystoreFile` a `keystorePass`. KeystoreFile udáva umiestnenie úložiska s certifikátom servera, keystorePass zase heslo do tohto úložiska (nepovolané osoby by nemali mať do úložiska certfikátov prístup). Cesta k certifikátu je buď absolútna alebo relatívna. V druhom prípade sa hľadá vzhľadom k hodnote premennej `CATALINA_HOME`. Certifikát z uvedeného súboru sa odosiela klientovi pri SSL handshakingu.

5.2 SSL spojenie medzi komponentmi

Medzi všetkými komponentmi systému je tiež možná bezpečná a šifrovaná komunikácia prostredníctvom SSL. Pre povolenie tejto komunikácie je potrebné urobiť niekoľko krokov.

Prvým je, aby servery, na ktorých beží `UserManager` a `PatientAdminInfo`, mali svoje certifikáty. Ich certifikáty nemusia byť vydané certifikačnou autoritou, stačí ich vygenerovať, napríklad použitím nástroja `keytool`, ktorý je súčasťou Java Development Kit.

Ak máme certifikáty, je nutné nejakým spôsobom oznámiť komponentu `Dispatcher`, aby dôveroval serverom, ktoré sa týmito certifikátmi preukážu. Preto musíme tieto certifikáty naimportovať do úložiska dôveryhodných certifikátov v rámci Java Runtime Enviroment, v ktorom beží aplikačný server (v našom prípade JRE 1.5 update 6). Tento import sa

prevedie použitím nástroja `keytool` (alebo ľubovoľným dostupným nástrojom na správu certifikátov) príkazom

```
>keytool -import -alias usermanager -file usermanager.cer
-keystore
c:\Program Files\Java\jre1.5_06\lib\security\cacerts
```

Prepínač `file` určuje súbor s certifikátom servera, na ktorom beží UserManager a prepínač `keystore` je úložisko dôveryhodných certifikátov JRE. Prepínač `alias` je identifikátor, pod ktorým je certifikát do úložiska vložený. Pri vkladaní certifikátu do uvedeného úložiska je potrebné zadať heslo úložiska, ktoré je v JRE nastavené na hodnotu „changeit“.

Podobný postup je nutné spraviť aj s certifikátom servera, na ktorom je spustený komponent `PatientAdminInfo`.

Na počítači, kde beží `Data Warehouse`, je nutné naimportovať podľa vyššie uvedeného postupu certifikát servera s komponentom `UserManager`. Je to nutné kvôli tomu, že `Data Warehouse` pred vložením dát autentizuje užívateľa a kontaktuje UserManagera.

Nastavenie bezpečnej komunikácie medzi komponentom Dispatcher a EHRClientom je popísané v predošlom texte.

5.3 SSL spojenie medzi komponentom PatientAdminInfo a LDAP serverom

Komponent `PatientAdminInfo` sa voči LDAP serveru chová ako klient a preto je nutné aj na počítači, kde beží tento komponent, naimportovať certifikát LDAP servera. Postup importu je uvedený vyššie.

Kapitola 6
Záver

Server bol testovaný na systéme, kde bol spustený aplikačný server aj adresárový server. Na aplikačnom serveri bežali všetky komponenty, takže boli postačujúce dva certifikáty – certifikát aplikačného a adresárového servera. Na priloženom CD sa nachádza úložisko s certifikátom, ktorým sa preukazuje aplikačný server (súbor `mudrserver`), úložisko s certifikátom, ktorým sa preukazuje adresárový server (súbor `ldap.keystore`) a samotné certifikáty `ldap.cer` (vyexportovaný certifikát z úložiska `ldap.keystore`) a `mudrj.cer` (vyexportovaný certifikát z úložiska `mudrserver`), ktoré je potrebné naimportovať k dôveryhodným certifikátom JRE podľa postupu uvedeného v predošlom texte.

Testovacia platforma bola Windows (Windows XP SP 2), bolo použité JRE 1.5 update 6, Axis2 verzia 1.2 a adresárový server Apache Directory Server 1.0.

Server elektronického zdravotného záznamu implementovaný v tejto práci bol úspešne nainštalovaný a otestovaný na platforme Win32. Server nejavil žiadne výrazné nedostatky. Bola ocenená jeho jednoduchá rozšíriteľnosť a prehľadná dekompozícia na viacero komponentov.

Do budúcna sa uvažuje o rozšírení servera elektronického zdravotného záznamu o kontrolu obmedzení hodnôt ukladaných v dátových stromoch. Na tento účel by bolo vhodné použiť princípy archetypov [11].

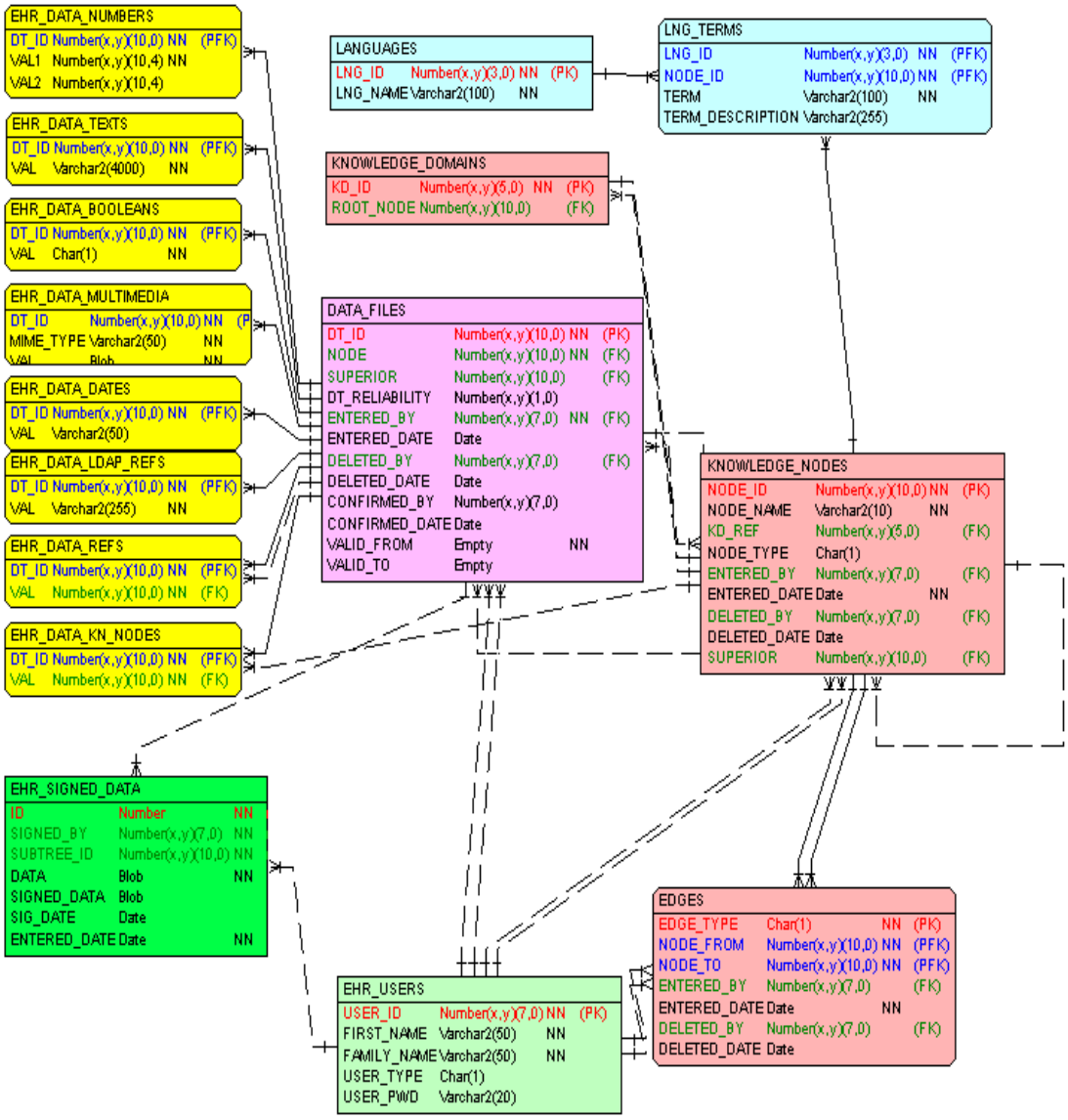
Prílohy

Príloha 1: Schéma LDAP stromu na ukladanie administratívnych údajov pacientov.

```
# patients
# Generated By LDAPStudio on : 4.4.2007 11:01:59

attributetype ( 1.2.3.4.5.6.20
      NAME 'birthDate'
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.53
      SINGLE-VALUE
 )

attributetype ( 1.2.3.4.5.6.19
      NAME 'deleted'
      DESC 'indikator, ci je pacient zmazany'
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
      SINGLE-VALUE
 )

attributetype ( 1.2.3.4.5.6.18
      NAME 'contactType'
      DESC 'typ kontaktu (pri otherContact)'
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.44
 )

attributetype ( 1.2.3.4.5.6.10
      NAME 'insuranceCompanyAttr'
      DESC 'DN poistovne, v ktorej je pacient poisteny (odkaz
do ou=insuranceComapnies,o=patients)'
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
      SINGLE-VALUE
 )

attributetype ( 1.2.3.4.5.6.11
      NAME 'sex'
      DESC 'pohlavie pacienta'
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
      SINGLE-VALUE
 )
attributetype ( 1.2.3.4.5.6.17
      NAME 'birthnum'
      DESC 'rodne cislo pacienta, bez lomitka'
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.44{11}
      SINGLE-VALUE
 )

attributetype ( 1.2.3.4.5.6.16
      NAME 'city'
```

```
      DESC 'mesto'
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.44
 )
objectclass ( 1.2.3.4.5.6.3
      NAME 'insuranceCompany'
      DESC 'poistovna'
      STRUCTURAL
      MUST ( uid $ description )
 )
objectclass ( 1.2.3.4.5.6.5
      NAME 'contact'
      DESC 'abstraktny kontakt - budu dedit triedy
HomeContact, WorkContact, PostalContact, OtherContact'
      ABSTRACT
      MUST cn
      MAY ( email $ postalCode $ telephoneNumber $ street $
city $ c $ st )
 )
objectclass ( 1.2.3.4.5.6.12
      NAME 'homeContact'
      DESC 'domaci kontakt na pacienta'
      SUP contact
      STRUCTURAL
 )
objectclass ( 1.2.3.4.5.6.13
      NAME 'workContact'
      DESC 'pracovny kontakt pacienta'
      SUP contact
      STRUCTURAL
 )
objectclass ( 1.2.3.4.5.6.1
      NAME 'patient'
      DESC 'obsahuje administrativne  zaznamy o pacientoch'
      STRUCTURAL
      MUST ( givenName $ sn $ uid )
      MAY ( sex $ birthnum $ insuranceCompanyAttr $ birthDate
$ deleted )
 )

objectclass ( 1.2.3.4.5.6.14
      NAME 'postalContact'
      DESC 'postovy kontakt na pacienta (p.o. box  a pod.)'
      SUP contact
      STRUCTURAL
      MAY postOfficeBox
 )
objectclass ( 1.2.3.4.5.6.15
      NAME 'otherContact'
      DESC 'iny kontakt'
      SUP contact
      STRUCTURAL
      MAY contactType
 )
```

Príloha 2: ER-schéma dátovej vrstvy, ktorá slúži na ukladanie klinických informácií pacientov



Literatúra

[1] Špidlen J.: *Databázová reprezentace medicínských informací a lékařských doporučení.* Diplomová práce, 2002, Matematicko-fyzikální fakulta, Univerzita Karlova v Praze.

[2] Kolesa P.: *Analýza a implementace aplikačního serveru pro elektronický zdravotní záznam.* Diplomová práce, 2004, Matematicko-fyzikální fakulta, Univerzita Karlova v Praze.

[3] Apache Software Foundation : *Axis2*, 2007, http://ws.apache.org/axis2/

[4] Apache Software Foundation: *Apache Tomcat*, 2007, http://tomcat.apache.org/index.html

[5] World Wide Web Consortium: *SOAP*, 2007, http://www.w3.org/TR/soap12-part1/

[6] World Wide Web Consortium: *XML Schema*, 2004, http://www.w3.org/XML/Schema

[7] World Wide Web Consortium: *WSDL*, 2001, http://www.w3.org/TR/wsdl

[8] Internet Engineering Task Force : *LDAP v3*, 1997 , http://www.ietf.org/rfc/rfc2251.txt

[9] Apache Software Foundation: *Apache Directory*, 2003-2007, http://directory.apache.org/apacheds/1.0/

[10] Internet Engineering Task Force: *LDIF*, 2000, http://www.ietf.org/rfc/rfc2849.txt

[11] The openEHR Foundation: *Archetype principles*, 2005 , http://svn.openehr.org/specification/TAGS/Release-1.0/publishing/architecture/am/archetype_principles.pdf

[12] Apache Software Foundation: *Apache Tomcat 6.0 SSL configration*, 2006, http://tomcat.apache.org/tomcat-6.0-doc/ssl-howto.html

[13] RSA Laboratories: *What is public-key cryptography?*, 2007, http://www.rsa.com/rsalabs/node.asp?id=2165

[14] Apache Software Foundation: *Apache Directory Server - Basic configuration tasks*, 2007, http://directory.apache.org/apacheds/1.0/14-basic-configuration-tasks.html#1.4.Basicconfigurationtasks-Addingyourownpartitionresp.suffix

[15] Oracle: *Oracle Database 10g*, 2007 , http://www.oracle.com/technology/products/database/oracle10g/index.html

[16] Object Management Group: *Person Identification Service Specification*, 2001, http://www.omg.org/docs/formal/01-04-04.pdf